EXPRESS MAIL LABEL NO:
EV222905803 US

Attorney Docket No.: QE1048.US

SYSTEM AND METHOD FOR CONCATENATING DATA

Angel G. Perozo

Theodore C. White

William W. Dennin

CROSS REFERENCE TO RELATED APPLICATIONS:

[0001] This patent application is related to and claims priority to U.S. provisional patent application Serial Number: 60/444,339, Filed on 01/31/2003, Docket Number QE1048.USPROV, entitled "System and Method for Coalescing Data", the disclosure of which is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field Of the Invention

[0002] The present invention relates generally to storage device controllers, and more particularly, to streamlining data flow in storage device controllers.

2. Background

[0003] Conventional computer systems typically include several functional components. These components may include a central processing unit (CPU), main memory, input/output ("I/O") devices, and streaming storage devices (for example, tape drives) (referred to herein as "storage device"). In conventional systems, the main memory is coupled to the

CPU via a system bus or a local memory bus. The main memory is used to provide the CPU access to data and/or program information that is stored in main memory at execution time. Typically, the main memory is composed of random access memory (RAM) circuits. A computer system with the CPU and main memory is often referred to as a host system.

[0004] The storage device is coupled to the host system via a controller that handles complex details of interfacing the storage devices to the host system. Communications between the host system and the controller is usually provided using one of a variety of standard I/O bus interfaces.

[0005] Typically, when data is read from a storage device, a host system sends a read command to the controller, which stores the read command into the buffer memory. Data is read from the device and stored in the buffer memory.

[0006] In storage devices, data is stored in blocks of varying sizes and non-contiguous segments. However, data, when sent to the host system must be contiguous. Therefore, data, after being read should be assembled efficiently, so that, to a host system it appears contiguous. Conventional systems do not perform this function efficiently.

[0007] Therefore, there is a need for a system to assemble data so that when it is sent to the host it is contiguous.

SUMMARY OF THE INVENTION

[0008] In one aspect of the present invention, a read assembly module in a storage controller for concatenating data segments is provided. The read assembly module includes, a first register for receiving data segments and a data start address; and a second register for concatenating data segments having different data segment size.

[0009] The read assembly module also includes a third register that holds data before it is sent to the second register; and a fourth register for padding data segments, if padding is needed.

[0010] In yet another aspect of the present invention, a system for concatenating data segments before the data segments are sent to a requesting host system is provided. The system includes, a read assembly module; a read command queue, which provides information to the read assembly module; and a controller that controls the read command queue.

[0011] In yet another aspect of the present invention, a method for concatenating data segments before the data segments are sent to a requesting host system is provided. The method includes, receiving

data start address; and assembling data segments such that the segments appear contiguous.

[0012] This brief summary has been provided so that the nature of the invention may be understood quickly. A more complete understanding of the invention can be obtained by reference to the following detailed description of the preferred embodiments thereof concerning the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The foregoing features and other features of the present invention will now be described with reference to the drawings of a preferred embodiment. In the drawings, the same components have the same reference numerals. The illustrated embodiment is intended to illustrate, but not to limit the invention. The drawings include the following Figures:

[0014] Figure 1A shows a block diagram of a controller, according to one aspect of the present invention;

[0015] Figure 1B is a block diagram of a buffer controller, according to one aspect of the present invention;

[0016] Figure 2 shows examples of using the read assembly unit, according to one aspect of the present invention;

[0017] Figure 3 is a block diagram of a read assembly unit, according to one aspect of the present invention;

[0018] Figure 4 shows a block diagram of a Channel interface used by the buffer controller of Figure 1B and uses the read assembly unit of Figure 3, according to one aspect of the present invention; and

[0019] Figure 5 is a block diagram of a read assembly command queue (Scatter Gather Module) interfacing with various other components, according to one aspect of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0020] To facilitate an understanding of the preferred embodiment, the general architecture and operation of a controller will initially be described. The specific architecture and operation of the preferred embodiment will then be described with reference to the general architecture.

[0021] The system of Figure 1A is an example of a streaming storage drive system (e.g., tape drive), included in (or coupled to) a computer system. The host computer (not shown) and the storage device 115 communicate via port 102, which is connected to a data bus (not shown). In an alternate embodiment (not shown), the storage device 115 is an external storage device, which is connected to the host computer via a

data bus. The data bus, for example, is a bus in accordance with a Small Computer System Interface (SCSI) specification. Those skilled in the art will appreciate that other communication buses known in the art can be used to transfer data between the drive and the host system.

[0022] As shown in Figure 1A, the system includes controller 101, which is coupled to SCSI port 102, port 114, buffer memory 111 and microprocessor 100. Interface 118 serves to couple microprocessor bus 107 to microprocessor 100. A read only memory ("ROM") omitted from the drawing is used to store firmware code executed by microprocessor 100. Port 114 couples controller 101 to device 115.

[0023] Controller 101 can be an integrated circuit (IC) that comprises of various functional modules, which provide for the writing and reading of data stored on storage device 115. Microprocessor 100 is coupled to controller 101 via interface 118 to facilitate transfer of data, address, timing and control information. Buffer memory 111 is coupled to controller 101 via ports to facilitate transfer of data, timing and address information.

[0024] Data flow controller 116 is connected to microprocessor bus 107 and to buffer controller 108. A DMA interface 112 is connected to microprocessor bus 107 and to data and control port 113.

[0025] SCSI controller 105 includes programmable registers and state machine sequencers that interface with SCSI port 102 on one side and to a fast, buffered direct memory access (DMA) channel on the other side.

[0026] Sequencer 106 supports customized SCSI sequences, for example, by means of a 256-location instruction memory that enables users to customize command automation features. Sequencer 106 is organized in accordance with the Harvard architecture, which has separate instruction and data memories. Sequencer 106 includes, for example, a 32-byte register file, a multi-level deep stack, an integer algorithmic logic unit (ALU) and other special purpose modules. Sequencer 106 supports firmware and hardware interrupts schemes. The firmware interrupt enables microprocessor 100 to initiate an operation within Sequencer 106 without stopping sequencer operation. Hardware interrupt comes directly from SCSI controller 105.

[0027] Buffer controller (also referred to as "BC") 108 connects buffer memory 111, DMA I/F 112, a SCSI channel of SCSI controller 105 and to bus 107. Buffer controller 108 regulates data movement into and out of buffer memory 111.

[0028] To read data from device 115, a host system sends a read command to controller 101, which stores the read commands in buffer memory 111. Microprocessor

100 then reads the command out of buffer memory 111 and initializes the various functional blocks of controller 101. Data is read from device 115 and is passed through DMA I/F 112 to buffer controller 108.

[0029] Figure 1B shows a block diagram of BC 108 with Channel 1 108A and Channel 0 108D interfaces for moving data to and from buffer 111. BC 108 includes register(s) 108E and an Arbiter 108C. Arbiter 108C arbitrates between plural channels in BC 108, for example, Channel 0 108D and Channel 1 108A. Register 108E is coupled to interface 118 via bus 107 that allows microprocessor 100 and BC 108 to communicate. Data 108G and status 108F is moved in and out of register 108E.

[0030] BC 108 also includes a memory controller 108B that interfaces with buffer 111 through a synchronous dynamic random access memory ("SDRAM") or dynamic random access memory ("DRAM") interface 108J.

[0031] Figure 4 shows a block diagram, of Channel 1 108A interface, which includes read assembly unit 403 and read assembly command queue 402 (may also be referred to as "scatter gather module 402") that are described below, according to one aspect of the present invention.

[0032] Channel 1 108A also includes a first- in first-out memory buffer ("FIFO") 404 that receives data

from SCSI interface 105. Channel 1 108A also has plural registers 400 that are operationally coupled to a controller 401 that is coupled to a buffer memory 111 via interface 108J. Controller 401 includes a state machine 501 (Figure 5) that monitors/controls queue 402, as described below.

[0033] Read Assembly Module:

[0034] Figure 3 shows a block diagram of read assembly unit 403 in BC 108, according to one aspect of the present invention. Read assembly operations in controller 101 use the read assembly module 403 and the Scatter-Gather module 402 to concatenate data segments to be written into channel 1 108A FIFO 404. Each data segment is defined by a scatter-gather entry for length and location (data start address) within external buffer memory 111 with the start address of the data block. As the data blocks containing the data segments are read, module 403 extracts the data segments and writes them into Channel 1 108A FIFO 404. Entire data blocks are read to verify data integrity by using the CRC (cyclic redundancy code). The data segments then appear contiguous to a host that had a read request and can include multiple data blocks. Also, multiple blocks are pieced together without loosing any data integrity because data length can span multiple blocks and when

data length crosses over CRC, the CRC is not transferred to FIFO 404.

[0035] In one embodiment of the present invention, read assembly module (or unit) 403 (may also be referred to as module 403 or unit 403) is a pipelined design. Buffer controller 108 control signals, address, data and masks are registered in register 301 of read assembly module 403. Data collection starts when a cycle includes the byte address contained in a data start address register 510 (Figure 5).

[0036] Register 301 receives address from register 510, data 311, mask, cyclic redundancy code ("CRC") and data acknowledgement (DACK_REG). The address denotes the location from where data is to be read.

[0037] A "shuttle" register 302 is used to hold data temporarily for later assembly in register 303. Although shuttle register 302 may be of any size, in one aspect of the present invention, it may be 7 bytes wide. Shuttle register 302 receives data from register 301.

[0038] "Data- Out" (or output) register 303 collects data from shuttle register 302 and register 301 via multiplexer ("Mux") shifter 304 for assembly. Mux shifter 304 selects the valid byte(s) from registers 301 and 302.

[0039] In one aspect, register 303 collects an 8-byte word. Channel 1 FIFO 404 may be written 8 bytes at a time, except for the last write, which may only be 4 bytes. A segment can be of any byte length supported by the Data Length counter/register (511, Figure 5), so it is possible to end up with a non-MOD4 residue in shuttle register 302.

[0040] A force transfer signal (FORCE_XFR) is used to push non-MOD4 residues into the FIFO 404. The non-MOD4 residue is padded (305A) to the nearest MOD4 boundary before it is written into the FIFO 404 via register 306. The padding 305A occurs in register 306.

[0041] Read assembly unit 403 also includes logic for two adders, 308 and 310. Logic 308 receives the number of new bytes from register 301 and the number of valid bytes 309 from register 302. Output 308A of logic 308 is sent to register 302.

[0042] Logic 310 receives the number of new bytes from register 301 and the number of valid bytes from register 302. Output 310A of logic 310 is sent to register 303.

[0043] Read Assembly Operations:

[0044] Read assembly operations concatenate data segments and write them into FIFO 404. Data segments reside in buffer 111 and are defined using a data length register/counter 511 and a data start address

register 510. The data length register 511 contains the segment length in bytes. The data start address register 510 contains the byte address from where the segment begins. The data start address can be loaded with any byte address within the BC 108-address space into register 301.

[0045] Read assembly module 403 sends to FIFO 404 the selected data segment, but the entire block where the segment resides is read so that CRC can confirm data integrity. CRC bytes are not written into FIFO 404 nor accounted for in data length counter 511. In one aspect of the present invention, channel 1 108A operations are in double-word multiples, and the read-tape-assembly module 403 may pad (305A) the last word written into FIFO 404.

[0046] Multiple segments are concatenated using the scatter-gather queue 402. Each scatter-gather entry includes all data required to define a segment.

[0047] Figure 5 shows a block diagram showing read command queue 402 interaction with various components, according to one aspect of the present invention. Read command queue 402 may have any number of queues. Figure 5 illustrates only 8 queues (0-7). Each queue, for example queue 0, has various entries, collectively shown as 507, which includes, last block bit, sector

size, the start address for a sector, the start address for the data and the data length.

[0048] Queue management logic 508 controls queue entries and loads queue 402 using a valid bit 502. Queue 402 is functionally coupled to a transfer counter 509, address counter 506, data start address register 510 and data length counter 511.

[0049] Transfer counter 509 counts the number of sectors that have been transferred. Address counter 506 maintains a count for buffer 111 addresses.

[0050] Data start address register 510 provides the address of the data to register 301 in read assembly unit 403 (shown as read assembly data path 504 in Figure 5). Data length counter 511 maintains the count of data that is moved, which terminates the operation after the last block.

[0051] Output 505 is used for diagnostic purposes and is sent to register 400. State machine 501 (in controller 401) interfaces with read assembly path 504 and queue 402.

[0052] Figure 6 shows a process flow diagram for concatenating data segments so that the data segments appear contiguous. The process starts in step S601. Register 301 receives data start address from register 510. In step S602, data segments are loaded in register 301.

[0053] In step S603, data assembly takes place in register 303 that receives the number of bytes from register 301 and valid bytes from shuttle register 302. If any padding 305A is required, then it is done in register 306.

[0054] Thereafter, in step S604, data concatenated data segments are moved into FIFO 404. The process stops based on the value of data length counter 511.

[0055] Figure 2 shows an example of how data is concatenated, using the foregoing adaptive aspects of the present invention. It is noteworthy that the example in Figure 2 is not intended to limit the invention, but is only to provide an example.

[0056] Figure 2 shows three data segments in buffer 111. Data block address is shown as 0, 8, 16, 24, 32 and 40 on one side and 7, 15, 23, 31, 39 and 47 on another side. Multiple data blocks can be read and concatenated, while maintaining data integrity. Hence, if a large block of data has to be transferred, for example 10MB, the data is broken in to chunks, for example, 512 bytes with 4 bytes of CRC. These blocks are then stored and read in segments, according to one aspect of the present invention.

[0057] The first data segment has a data block address of 0, data start address of 3, length 3 bytes and block size 16. This segment is copied into FIFO

404. The second data segment has a length of 1 byte, a start address 16 and data block address of 16. The third segment also has a data block address of 16, length of 7 bytes, and data start address of 27. All this information is sent to register 301.

[0058] All three data segments are concatenated and written into FIFO 404 so that they appear contiguous to a requesting host system (not shown). Also, FIFO 404 may have three (3) pad bytes because the total length (addition of the three segments) was not mod4 and the last segment goes across a block boundary operation. It is noteworthy that the CRC is not transferred to FIFO 404. As shown in Figure 2, multiple blocks of data from buffer memory 111 are concatenated together, while data integrity is maintained.

[0059] In one aspect of the present invention, data fragments are efficiently concatenated.

[0060] Although the present invention has been described with reference to specific embodiments, these embodiments are illustrative only and not limiting. Many other applications and embodiments of the present invention will be apparent in light of this disclosure.